Solving Games using the combination of Q-learning and Regret Matching Methods

Sourav Chakraborty, Nagarajan Shanmuganathan

University of Colorado Boulder
sourav.chakraborty@colorado.edu
nagarajan.shanmuganathan@colorado.edu

Abstract. It is known well that Counterfactual regret minimization (CFR) has been used in games which have both terminal states and perfect recall to minimize regret. This project aims to relax those constraints and use a local no-regret algorithm (LONR) which internally uses a Q-learning like update rule to games which do not have terminal states or perfect recall. This project aims to implement the LONR algorithm and test it on various kinds of games and see report the convergence results.

Keywords: Q-Learning, No regret algorithms, NoSDE Games

1 Introduction

Counterfactual regret minimization (CFR) has been found to solve extensive games with incomplete information. But it does have some limitations. It makes two strong assumptions: The first assumption is that the agents have *perfect recall*, which means that the state representation captures the full history of the states visited. Second, CFR performs updates only after a terminal state is reached, which means that the games that we consider need to have terminal states.

When we consider the scenarios from the reinforcement learning (RL) domain, it is not uncommon for the setting to not encode the full history of states and actions, since the state space that we consider will be really large. Also, even if the RL settings do have some terminal states, it might take a lot of time steps to even reach one of those states and it's not efficient in terms of storage to have the entire computation in memory.

These reasons motivate us to use learning algorithms for these settings which are usually called "online Markov decision processes" (MDP). The remainder of this report covers some preliminaries, the main idea behind a novel local no-regret (LONR) algorithm [1], the convergence result of LONR, the implementation details of the algorithm and the convergence results.

2 Preliminaries

This section discusses the basic definitions and brief explanation of the concepts that are used in the project.

2.1 Definitions

A Markov Decision Process $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space which is finite, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability kernel, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the expected reward function, and $0 < \gamma < 1$ is the discount rate.

The Q-value iteration is an operator $\mathcal{T}$, whose domain is bounded real-valued functions over $\mathcal{S} \times \mathcal{A}$, defined as,

$$\mathcal{T}\mathcal{Q}(s,a) = r(s,a) + \gamma \mathbb{E}_P[max_{a' \in \mathcal{A}} \mathcal{Q}(s', a')]]$$

Since most of the work of this project deals with regret minimization, let's define mathematically how to find the regret of an algorithm. Consider a setting where there are n actions $a_1, \ldots a_n$. At each time step k an online algorithm chooses a probability distribution π_k over the n actions. Then an adversary chooses a reward $x_{k,i}$ for each action i from some closed interval [0, 1], which the algorithm then observes. The regret of the algorithm at time k is

$$\frac{1}{k+1} max_i \sum_{t=0}^{k} x_{t,i} - \pi_t . x_t$$

An algorithm is *no-regret* if there is a sequence of constants ρ_k such that regardless of the adversary the regret at time k is at most ρ_k and $lim_{k \to \infty}\ \rho_k = 0$. A common bound is ρ_k is $\mathcal{O}(1 \backslash \sqrt{k})$.

An algorithm is *no-absolute-regret*, if the *absolute value* of the regret is bounded by ρ_k.

2.2 Classes of Games and Some Examples

In order to deal with multi agent settings, games are usually modeled as Markov games, which are a generalization of MDPs to multi-agent settings. A Markov Game Γ is a tuple $(S, N, \mathbf{A}, T, R, \gamma)$ where S is the set of states, $N = 1, \ldots, n$ is the set of players, the set of all state-action pairs $\mathbf{A} = \bigcup_{s \in S} (s \times \prod_{n \in N} A_{n,s})$, a transition kernel $T : \mathbf{A} \mapsto \Delta(S)$, and a discount factor γ.

In particular, we are interested in NoSDE Markov Games, which pose good challenges to learning algorithms. These are finite two agent Markov games with no terminal states where No Stationary Deterministic Equilibria exist: all stationary equilibria are randomized.

Grid World The Grid World [2] is a *cliff walking* setting. It is an undiscounted, episodic navigating task where in which the agent find its way from start to goal in a deterministic grid world. Along the edge of the grid world is a cliff (Fig. 1).The agent can take any of four movement actions: *north, south, east* and west, each of which moves the agent one square in the corresponding direction. Each step results in a reward of -1, except when the agent steps into the cliff area, which results in a reward of -100 and an immediate return to the start state. The episode ends upon reaching the goal state.

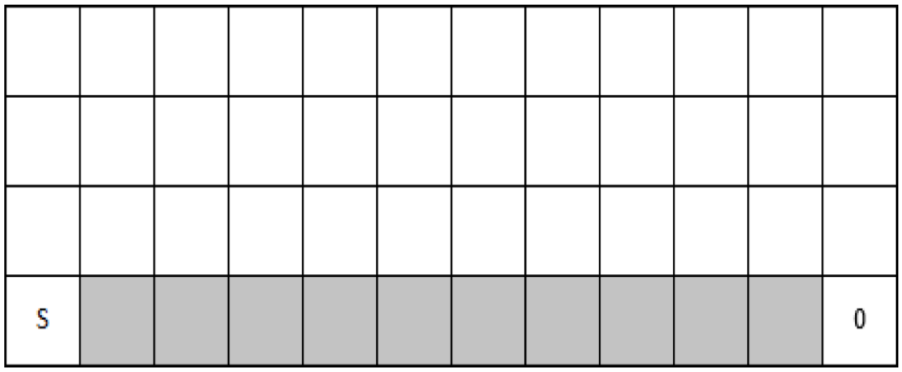

Fig. 1. The Grid World Setting

More details about the experiment set up and the parameters of this game will be discussed in the Experiments section.

This is a one player game and for our current and future experiments, we also will like to get results from two player games such as Rock-Paper-Scissors and Soccer game.

Soccer Game The game [3] is played on a 4x5 grid as depicted in Fig. 2. The two players, A and B, occupy distinct squares of the grid and can choose one of 5 actions on each turn: N, S, E, W, and stand. Once both players have selected their actions, the two moves are executed in random order.

The circle in the figures represents the "ball." When the player with the ball steps into the appropriate goal (left for A, right for B), that player scores a point and the board is reset to the configuration shown in the left half of the figure. Possession of the ball goes to one or the other player at random.

When a player executes an action that would take it to the square occupied by the other player, possession of the ball goes to the stationary player and the move does not take place. A good defensive maneuver, then, is to stand where the other player wants to go. Goals are worth one point and the discount factor $(0 < \alpha < 1)$ is set , which makes scoring sooner somewhat better than scoring later.

For an agent on the offensive to do better than breaking even against an unknown defender, the agent must use a probabilistic policy. For instance, in the example situation shown in the right half of Fig. 2, any deterministic choice for A can be blocked indefinitely by a clever opponent. Only by choosing randomly between stand and S can the agent guarantee an opening and therefore an opportunity to score.

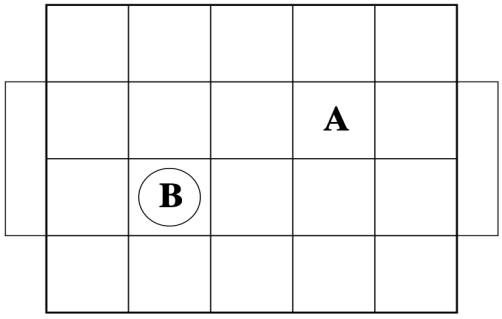

Fig. 2. Soccer Game Setting

3 LONR Algorithm

LONR, as proposed in [1] has the same asymptotic convergence guarantee as value iteration for discounted-reward MDPs. Both CFR and LONR are guaranteed to converge only in terms of their average policy. This is part of a general phenomenon for no-regret learning in games, where the "last iterate," or current policy, not only fails to converge but behaves in an extreme and cyclic way.

3.1 Algorithm Overview

The idea of LONR is to fuse the essence of value iteration / Q-learning and CFR. A standard analysis of value iteration proceeds by analyzing the sequence of matrices $\mathcal{Q}, \mathcal{T}\mathcal{Q}, \mathcal{T}^2\mathcal{Q}, \mathcal{T}^3\mathcal{Q}, \ldots$. CFR chooses the policy for each state locally using a no-regret algorithm which yields a sequence of Q matrices as follows.

Find a matrix Q_0. Initialize $|\mathcal{S}|$ copies of a no-absolute-regret algorithm with $n = |\mathcal{A}|$ and find the initial policy $\pi_0(s)$ for each state s. Then iteratively reveal rewards to the copy of the algorithm for state s as $x^s_{k,i} = Q_k(s, a_i)$ and update the policy π_{k+1} according to the no-absolute-regret algorithm and $Q_{k+1}(s, a) = r(s, a) + \gamma \mathbb{E}_{P,\pi_k}[Q_k(s', a')]$.

It has been shown in [1] through a series of lemmas that LONR converges to Q^*, the optimal policy and the convergence is of the average of the Q_k matrices.

The main result is stated here:

Theorem 1. $lim_{k\to\infty} Q_k = Q^*$

3.2 RM++

One of the main functionalities of the LONR algorithm is the *policy update* part where it updates the current policy at every iteration. For this step, traditionally in the literature Regret Matching (RM) algorithms are used. These algorithms

are the widely used regret minimizers in CFR-based algorithms. With RM, the policy distribution for iteration $t+1$ is selected for actions proportional to the accumulated positive regrets over iterations 0 to t.

Regret Matching+ (RM+) is a variation that resets negative accumulated regret sums after each iteration to zero, and applies a linear weighing term to the contributions to the average strategy.

In [1], a new variant of RM called Regret Matching++ (RM++) is introduced which updates in a similar fashion to RM but clips the instantaneous regrets at 0. It is also proved that RM++ is a no-regret algorithm and also that RM++ empirically has last iterate convergence in various settings. This is the reason why we are interested to use this setting for different classes of games and check how well RM++ can be used in such games.

3.3 LONR Pseudocode

The main two steps Q-update and Policy update as given in [1] is given here. The way those two have been implemented will be discussed in the Experiments section.

procedure Q-UPDATE(n, s, t) ▷ Update Q-Values
  for each action $a_n \in A_n(s)$ **do**
    $successors = getSuccessorStatesAndTransitionProbs(n, s, a_n, a_{-n})$
    $ActionValue \leftarrow 0$
    for s', $transProb$, $reward$ in $successors$ **do**
      $nextStateValue \leftarrow \sum_{a'_n} Q_t(n, s', a'_n) \times \pi_t(n, s', a'_n)$
      $ActionValue \leftarrow ActionValue + transProb \cdot (reward + \gamma \cdot nextStateValue)$
    $Q_{t+1}(n, s, a_n) \leftarrow ActionValue$

procedure POLICY-UPDATE(n, s, t) ▷ Regret Matching++
  $ExpectedValue = \sum_{a_n} Q_{t+1}(n, s, a_n) \times \pi_t(n, s, a_n)$

  for $a_n \in A_n(s)$ **do** ▷ RM++ Update Rule
    $immediateRegret \leftarrow max(0, Q_{t+1}(n, s, a_n) - ExpectedValue)$
    RegretSums$(n, s, a_n) \leftarrow$ RegretSums(n, s, a_n) + $immediateRegret$

  $totalRegretSum = \sum_i$RegretSums(n, s, i)

  for $a_n \in A_n(s)$ **do** ▷ Update Policy
    if $totalRegretSum > 0$ **then**
      $\pi_{t+1}(n, s, a_n) = \frac{RegretSums(n,s,a_n)}{totalRegretSum}$
    else
      $\pi_{t+1}(n, s, a_n) = \frac{1}{|A_n(s)|}$

    PolicySums$(n, s, a_n) \leftarrow$ PolicySums$(n, s, a_n) + \pi_{t+1}(n, s, a_n)$

Fig. 3. Q-Update and Policy-Update in LONR

4 Implementation

This section will deal with the implementation details [4] of the *LONR* and about how the games are modeled as MDPs with their parameters.

For the project, the algorithms were implemented and tested using Python. It has been done in a way that new games can be added without modifying the existing implementation of LONR and other games.

4.1 LONR

LONR as an algorithm works independent of the games that are being considered. The implementation has been done in a manner where the game needs to be modelled as an MDP with specific parameters that LONR expects and that can be fed to the LONR to see the convergence results. This is called the *environment* that LONR expects. The four parameters that LONR expects are:

- **actions**: (agent $\times$ state) $\rightarrow A \subseteq \mathcal{A}$
- **next_states**: (agent $\times$ state) $\rightarrow S \subseteq \mathcal{S}$
- **transitions**: (agent $\times$ initial_state $\times$ action $\times$ next_state) $\rightarrow p \in [0, 1]$
- **rewards**: (agent $\times$ state $\times$ action) $\rightarrow \mathbb{R}$

The implementation details of the LONR algorithm can be found at [4] /lonr/local_no_regret.py file.

4.2 Grid World

Our primary experiments are done on the Grid world game as described in Section 2.2. The grid size can be customized with different rows and columns to test out. The default values have been set to 4 rows with 12 columns and a cliff at the bottom row excluding the corner two cells which are the start and destination cells for the agent. The rewards at each time step has been modelled exactly as mentioned in Section 2.2. The implementation details of this can be found at [4] at /environments/gridworld.py.

4.3 Rock-Paper-Scissors

This is our attempt to make LONR work for two player zero sum games. If a round of the game ends in a tie, each player gets a reward of 0 otherwise, the winner gets a reward of 1 and the loser gets a reward of -1. There is only one state in this setting and the game resets to the same state after every round. This is still work in progress and the implementation details can be found at [4] at /environments/rps.py.

5 Experiments

This section discusses the experiments that we have performed with the LONR algorithm. We experimented with the Grid World environment and observed the convergence of the optimal value of the action *north* when the agent is at the starting position in Fig. 4. We also observed the convergence of the optimal policy for the single agent in the current environment in Fig. 5. We did that by finding the difference between the policy matrix at time t with the policy matrix at time $t-1$ and plotted the difference at every t. After few epochs it is evident that the policy does not change and the value suggests that it is the optimal one.

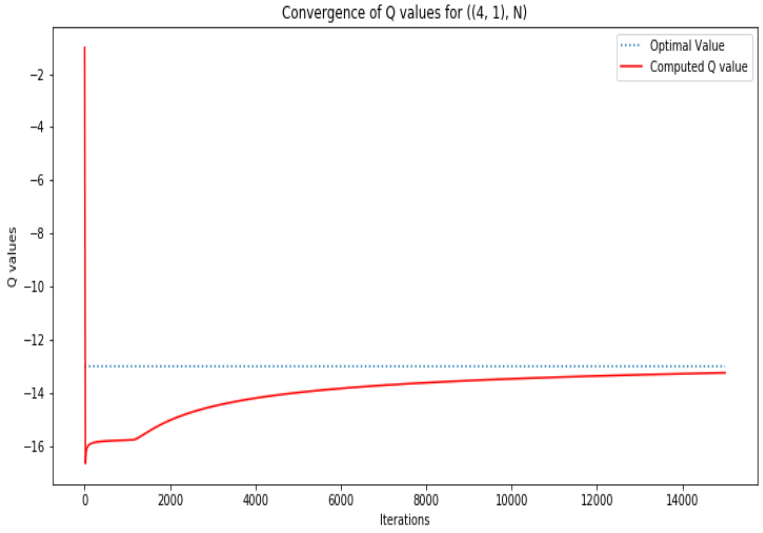

Fig. 4. Optimal Q value for the (Start, North) pair.

5.1 Running the programs

The experiments can be run either by running the `experiments.ipynb` file or `test.py` file by issuing the command `python3 test.py`. Upon running the `test.py`, we can see the converge result for the optimal value and also the optimal path gets printed.

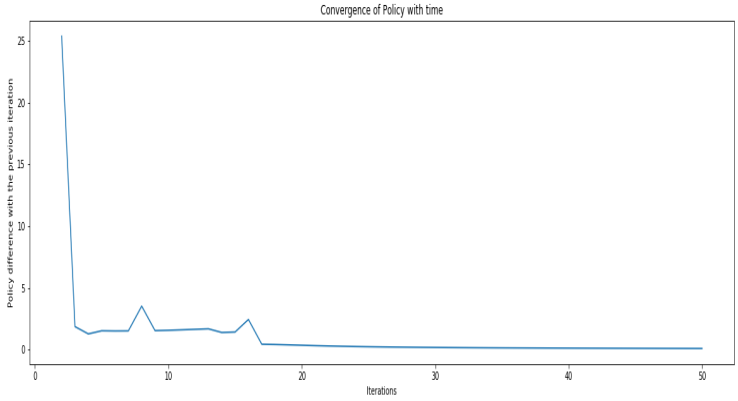

Fig. 5. Convergence of optimal policy

6 Further Readings and Research Directions

Apart from trying out different classes of games, there are various avenues to explore and we would like to continue the research work in some of these interesting areas. It would be interesting to see the effect of using Union Confidence Bounds for the Q learning portion. We also would like to explore more about the asynchronous bandit like settings. Another exciting area would be to come up with a new Regret Matching policy update algorithm.

7 Conclusion

It is always interesting to relate multiple concepts and use those to solve problems. This project gave us the insights of how we can use reinforcement learning to game settings. While the challenges are aplenty, the results that are obtained from using these techniques promises to open up a wide variety of research and it will be exciting to see how different game settings can be incorporated into the RL framework.

References

1. Ian A. Kash, Michael Sullins, Katja Hoffmann: Combining No-regret and Q-learning. (2019)
2. H. van Seijen, H. van Hasselt, S. Whiteson and M. Wiering, "A theoretical and empirical analysis of Expected Sarsa," 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, Nashville, TN, 2009, pp. 177-184.
3. Michael L. Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In Proceedings of the Eleventh International Conference on International Conference on Machine Learning (ICML'94). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 157–163.

4. https://github.com/souravchk25/alg-game-theory-project